RT Embedded http://www.kontronn.com

GPU实现的高速FIR数字滤波算法

陈孝良[1)] ,邓仰东[2)] ,程晓斌[1)] ,李晓东[1)] ,田 静[1)]

[1)] (中国科学院噪声与振动重点实验室(声学研究所) 北京 100190)

[2)] (清华大学微电子学研究所 北京 100084)

(cxl @mail.ioa.ac.cn)

摘 要:针对目前基于GPU的FIR算法速度低、扩展性差的缺点,提出一种高速的多通道FIR数字滤波的并行算法,并利用平衡并行运算负载的技术以及降低内存访问密度的方法进行加速.该算法采用矩阵乘法的并行运算技术在GPU上建立并行滤波模型,通过每个线程在单个指令周期内执行2个信号运算,实现了多通道信号的高速滤波.实验结果表明,在GTX260+平台上,采用文中算法的平均加速比达到了203,效率超过40%,并且具有更好的扩展性.

关键词:有限脉冲响应;数字滤波;并行计算;CUDA;GPU

中图法分类号: TP391

High Speed FIR Digital Filtering on GPU

Chen Xiaoliang[1)] , Deng Yangdong[2)] , Cheng Xiaobin[1)] , Li Xiaodong[1)] , and Tian Jing[1)]

[1)] (*Key Laboratory of Noise and Vibration Research, Institute of Acoustics, Chinese Academy of Sciences, Beijing* 100190)

[2)] (*Institute of Microelectronics, Tsinghua University, Beijing* 100084)

Abstract: This paper proposes a massively parallel FIR filter algorithm with its GPU implementation to improve the efficiency and scalability of GPU based FIR. By the algorithm proposed, the problem is formulated as a matrix multiplication operation that offers sufficient data level parallelism for parallel filtering on modern GPUs. In addition, the GPU implementation guarantees that each thread could complete a two-signal operation within a single instruction cycle. Efficient and effective strategies for load balancing and memory mapping are also introduced to further improve the performance. The proposed algorithm and the corresponding GPU implementation could achieve an efficient multi-channel signal filtering. Experimental results on a GTX260+ graphics card prove that the FIR filter algorithm could be used to deliver on average a speed-up of 203X and an efficiency increase over 40%.

Key words: finite impulse response (FIR); digital filter; parallel computing; CUDA; GPU

有限脉冲响应(finite impulse response,FIR)数字滤波器由于具有良好的稳定性和严格的线性相位特性,在语音和图像处理、多媒体应用、模式识别中具有广泛的应用[1],特别是在多通道信号分析和高质量图像处理中,要求高阶FIR数字滤波器具有更快的运算速度.随着计算机技术的发展,很多FIR优化算法已不再适用于新的计算机结构,因此,如何优化FIR数字滤波器的性能一直是人们研究并关注的热点问题.近几年的研究工作主要集中在VLSI,DSP和FPGA领域,提出的优化方法主要有2类:1)在时域降低FIR算法的复杂度[1-5];2)在Z域扩展FIR算法的并行性[6-9].

Created in Master PDF Editor

RT Embedded http://www.kontronn.com

第 1 类方法的典型算法有环形缓冲算法[1]、共享乘算法[2]、公共子表达式消去(common subexpression elimination, CSE)算法[3]、二元子表达式消去(binary subexpression elimination, BSE)算法[4]和资源优化算法[5]. BSE 算法是一种优化的 CSE 算法,其基于规范符号编码(canonical signed digit, CSD)对滤波器系数进行编码,通过移位和加法来降低乘法的数量,能够在一个时钟周期内完成一个激励信号与几个滤波器系数的乘法,在 VLSI 设计中能够获得很好的性能提升,并且有助于降低成本. 资源优化算法是一种分布式算法,由于涉及查找表的操作,仅适合于 FPGA 平台,比普通算法能够优化 29.7 %的 FPGA 单元[5]. 但是第 1 类算法都基于串行模式,受到芯片工作频率的制约,因此极难获得大幅度性能提升.

第 2 类方法的算法主要有快速 FIR(fast FIR algorithms, FFA)算法[6]、短时卷积算法[7]、余量算法[8]和两段并行算法[9]. 这类方法主要通过扩展 FIR 数字滤波器的吞吐量来提高总体运算性能. 但是 Z 域并行算法始终受到硬件成本的严重制约,不能同时支持较多的并发处理,获得更高加速比的硬件成本是线性增加的.

随着通用 GPU 技术的普及[10],人们提出第 3 种优化 FIR 的方法,即在时域扩展 FIR 算法的并行性[11-12]. 这类方法主要得益于 GPU 的低成本、细粒度和高并行性,能够大幅提升 FIR 数字滤波的性能. Koon 提出的 CUDARealFIR 算法[12]仅采用一块 GeForce 8800GTS 的显卡就能够带来最高 140 倍的加速比. 但是 GPU 的并行模式比较复杂,文献[1-9]中的优化算法都不适合直接移植,运行在 GPU 的算法需要重新设计并优化. 在 GPU 实现 FIR 数字滤波方面,Smirnov 等[11]首先进行了研究,CUDARealFIR 算法[12]是在文献[11]的基础上进行的优化,但是这 2 种算法存在许多限制,扩展性差,而且并行效率也不高.

针对上述 3 类方法在并行计算方面的不足,本文在分析通用 GPU 的并行架构和运行机制的基础上,从 FIR 算法的结构入手,提出了一种多通道并行滤波的高速算法 ——GPU HsFIR 算法. 该算法利用了现代通用 GPU 的并行优势,能够有效地实现 GPU 并行结构和内存系统的性能优化,从而提高了多通道滤波的速度,并且扩展了算法的应用范围.

1 基于卷积结构的 FIR 算法

FIR 数字滤波器在时域的卷积定义为

$$y(n) = \sum_{i=-\infty}^{\infty} h_i x(n-i) = \sum_{i=0}^{M} b_i x(n-i) \qquad (1)$$

其中,y 是响应信号,x 是激励信号,M 是滤波器长度,h_i 是脉冲响应,b_i 是滤波器系数,且 $b_i \neq 0$.

文献[11-12]提出的算法没有考虑 FIR 数字滤波器的瞬态响应过程,而是将式(1)直接改写成

$$y(n) = \sum_{i=0}^{M} b_i x(n+i) \qquad (2)$$

依据式(2),设 p 为线程数量,p_j 为第 j 线程,激励信号 x 长度为 $n \times p$,b_i 是滤波器系数,k 表示第 k 次迭代($k = 0,1,2,\cdots$),则输出响应

$$y(k \times p + p_j) = \sum_{i=0}^{M} b_i x(i + k \times p + p_j) \qquad (3)$$

Smirnov 等[11]依据式(3)首次采用 Cg 语言在 GeForce 6600 上实现了 FIR 数字滤波器的算法,激励信号 x 存储在纹理存储器中,但是算法性能并没有得到提升,甚至慢于 CPU 的算法. CUDARealFIR 算法[12]采用 CUDA 语言进行了优化,并且将其扩展到支持对 4 路 8 通道的音频信号处理. 虽然 CUDARealFIR 算法在性能方面获得了较大提升,但是仍然存在以下几个限制:

1) 式(2)并不是常用的 FIR 卷积结构,而很多应用需要分析瞬态响应过程,诸如 Matlab 的 Filter 函数则是标准的卷积结构;

2) 仅支持单通道滤波,只能通过循环调度该算法才能实现多通道滤波的功能;

3) 要求滤波器长度 $M \geq 1024$.

由于以上的限制,CUDARealFIR 算法的性能损失也是比较明显的. 根据式(3),若 $M > p$,则需要多读取 $n \times (M - p)$ 次全局存储器,而这正是 GPU 并行性能最大的瓶颈[13]. 另外,由于多通道滤波需要多次循环调度该算法,CPU —GPU 数据传输时间也是很大的瓶颈. 若设每通道输入 1 M 激励信号,GPU 内核加载时间 T_k,CPU —GPU 通过 PCIe 1.1 ×16 的传输时间为 T_t 则每通道滤波额外增加的时间开销[14]为

$$T_l = T_k + T_t = 7 + 11 + \frac{1\text{M}}{3.3\ \text{GB/S}} = 321\ \mu\text{s}.$$

显然,基于卷积结构在 GPU 上继续进行优化同样会面临上述问题,而且仅凭 GPU 编程技术的优化很难再获得更好的性能提升.

2 基于矩阵结构的并行算法

设 N 表示激励信号的长度,M 表示滤波器的长

RT Embedded http://www.kontronn.com

度, L 表示输入通道的数量,根据式(1),如果将激励信号 $x(n)$ 和响应信号 $y(n)$ 分别写成列向量

$$X=\begin{pmatrix}x_0\\x_1\\x_2\\x_3\\\cdots\\x_N\end{pmatrix},\quad Y=\begin{pmatrix}y_0\\y_1\\y_2\\y_3\\\cdots\\y_N\end{pmatrix};$$

设 $N>M$, b_i 是矩阵 $B_{N\times N}$ 中的元素

$$B=\begin{pmatrix}b_0&0&0&0&\cdots&0\\b_1&b_0&0&0&0&\cdots\\\cdots&b_1&b_0&0&0&0\\b_M&\cdots&b_1&b_0&0&0\\\cdots&b_M&\cdots&b_1&b_0&0\\0&\cdots&b_M&\cdots&b_1&b_0\end{pmatrix},$$

则式(1)可以写成 $Y=BX$ 即

$$\begin{pmatrix}y_0\\y_1\\y_2\\y_3\\\cdots\\y_N\end{pmatrix}=\begin{pmatrix}b_0&0&0&0&\cdots&0\\b_1&b_0&0&0&0&\cdots\\\cdots&b_1&b_0&0&0&0\\b_M&\cdots&b_1&b_0&0&0\\\cdots&b_M&\cdots&b_1&b_0&0\\0&\cdots&b_M&\cdots&b_1&b_0\end{pmatrix}\begin{pmatrix}x_0\\x_1\\x_2\\x_3\\\cdots\\x_N\end{pmatrix}.$$

若 L 通道激励信号的列向量定义为 $X_0, X_1, X_2, \cdots, X_L$;对应响应信号定义为列向量 $Y_0, Y_1, Y_2, \cdots, Y_L$;那么 $(Y_0, Y_1, Y_2, \cdots, Y_L)=B(X_0, X_1, X_2, \cdots, X_L)$,即

$$\begin{pmatrix}y_{00}&y_{10}&y_{20}&\cdots&y_{L0}\\y_{01}&y_{11}&y_{21}&\cdots&y_{L1}\\y_{02}&y_{12}&y_{22}&\cdots&y_{L2}\\y_{03}&y_{13}&y_{23}&\cdots&y_{L3}\\\cdots&&&&\cdots\\y_{0N}&y_{1N}&y_{2N}&\cdots&y_{Ln}\end{pmatrix}=\begin{pmatrix}b_0&0&0&0&\cdots&0\\b_1&b_0&0&0&0&\cdots\\\cdots&b_1&b_0&0&0&0\\b_M&\cdots&b_1&b_0&0&0\\\cdots&b_M&\cdots&b_1&b_0&0\\0&\cdots&b_M&\cdots&b_1&b_0\end{pmatrix}\begin{pmatrix}x_{00}&x_{10}&x_{20}&\cdots&x_{L0}\\x_{01}&x_{11}&x_{21}&\cdots&x_{L1}\\x_{02}&x_{12}&x_{22}&\cdots&x_{L2}\\x_{03}&x_{13}&x_{23}&\cdots&x_{L3}\\\cdots&&&&\cdots\\x_{0N}&x_{1N}&x_{2N}&\cdots&x_{Ln}\end{pmatrix}\qquad(4)$$

因此,FIR 数字滤波器可以从卷积结构转换成矩阵结构.本文提出的 GPU HsFIR 算法就是基于矩阵结构,它不仅消除了 CUDARealFIR 等算法的限制,而且可以借鉴并行计算中的 Cannon 算法和 DNS 算法[15],从而获得更高的并行性能. GPU HsFIR 算法相比 CUDARealFIR 算法的性能提升是非常显著的,图 1 表明,平均并行执行时间缩短了将近 50 倍.但是当 $N\geq 2048$ 时,两者之间的差距越来越小.这是因为,矩阵结构中 B 矩阵的尺寸是 N^2, N 变大后成为影响并行性能的主要因素.因此, GPU HsFIR 算法进一步考虑了并行计算模型的优化和内存映射策略的优化.

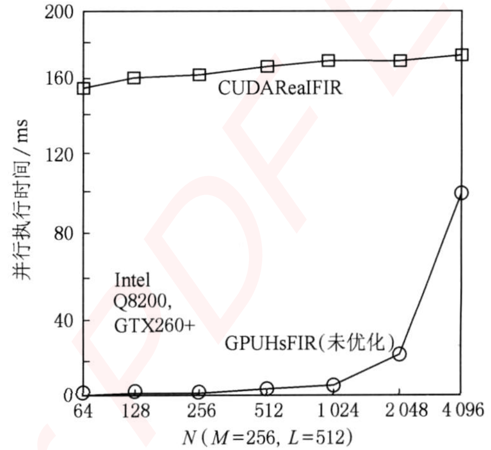


图 1 未优化的 GPU HsFIR 算法与 CUDARealFIR 算法比较

2.1 并行计算模型

CUDA 的并行计算模型由线程、线程块和网格组成,分别支持一维、二维和三维的线程组织.线程是 CUDA 语言的基本单位,具有独立的寄存器空间;线程块是一组互相合作的线程,可以通过共享存储器交换数据.线程块被拆分为若干 WARP,每个 WARP 则包含 32 个线程,在每一指令时钟周期上 WARP 内所有线程执行同一指令,但其针对不同的数据;而网格则是一组线程块,这些线程只能共享全局存储器、常数存储器和纹理存储器.

基于矩阵结构的 GPU HsFIR 算法适合二维结构的线程模型.图 2 所示为优化的并行计算模型,该模型首先将矩阵 B 和矩阵 X 分割为多个维度是 *block_size* 的子矩阵,然后通过这些子矩阵的乘累加来得到 Y_{sub},进而求得整个 Y 矩阵.其中,维度为 $(N, block_size)$ 的 B 子阵,行索引与 Y_{sub} 相同;维度为 $(block_size, N)$ 的 X 子阵,列索引与 Y_{sub} 相同.具体步骤如下:

Step1. 矩阵 B_M 首先存入常数存储器,矩阵 X 和 Y 存入全局存储器.

Step2. 每个线程块均负责计算 Y 的一个子矩阵 Y_{sub}.

Step3. 线程块内的每个线程通过共享存储器每次读入 X 的 2 个数据.

Step4. 线程块内的每个线程每次通过寄存器读取 B_M 的 2 个数据,并将其转换成 B 的数据.

Step5. 线程块内的每个线程负责计算 Y_{sub} 的 2 个数据,并将其直接输出到全局存储器.

Created in Master PDF Editor

RT Embedded http://www.kontronn.com

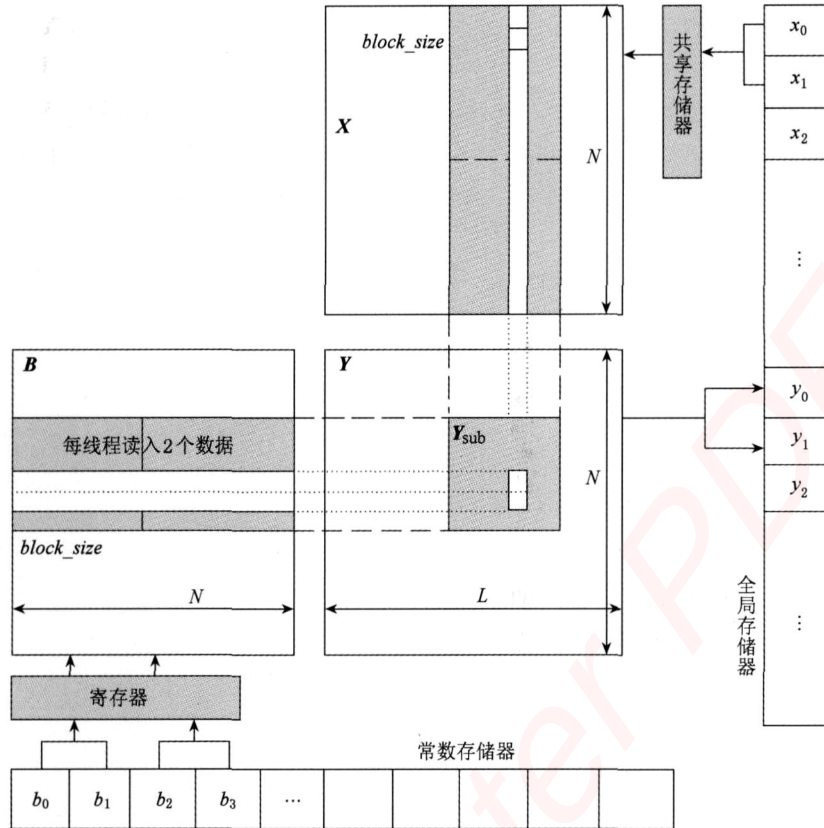


图 2　GPU HsFIR 算法的并行计算模型

2.2　内存映射策略

矩阵 B 是一种特殊的 Toeplitz 矩阵,并行计算中专门研究过这类矩阵的优化算法[16-17],但是这类算法存在内存访问频繁和算法分支过多的问题,而这都是 GPU 性能的瓶颈因素[14]. 由于这 2 个因素又互相制约,只能取其平衡,在减少算法分支的同时尽量减少内存访问次数[18]. 具体方法如下:

根据式(4),若定义 B_M 为一行向量 $B_M = b_0, b_1, b_2, b_3, \cdots, b_M$,则 B 中的任一元素 b_{ij} 可表示为

$$b_{ij} = \begin{cases} B_M(i - j), & 0 \le i - j \le M \\ 0, & i - j < 0, i - j > M \end{cases} \quad (5)$$

因此,B_M 向量可以完整地表示 B 矩阵,如果将其直接存储到 GPU 的常数存储器,利用常数存储器高速缓冲的特性,能够有效地提高 GPU 内存访问的效率. 图 3 所示的实验结果证明了这一点,优化后的 CPU —GPU 内存交换时间降到 10 ms 以下,并且传输速度比较稳定.

并行计算模型中每线程读取 2 个数据的方法不仅增加了数据读取的位宽(128 位),提升了内存访问效率,而且也减少了线程内算法分支的问题. GPU HsFIR 算法的内存映射分支代码片段如下:

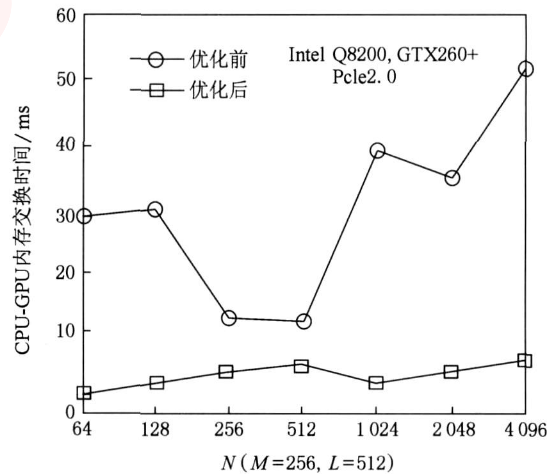


图 3　GPU HsFIR 算法内存映射策略优化的效果

```
_constant__float B[M]
_shared__float Bm[8][128]
for(int k=0;k<N;k +=128)
{
  __syncthreads();
  for(int u=0;u<8;u ++)
  {
    int b_y=threadIdx.y *8+u;
    int b_x=threadIdx.x *2;
    int B_y=blockIdx.y *8+b_y;
```

RT Embedded http://www.kontronn.com

```
      int B_x = k + b_x;
      int B_i = B_y - B_x;
      if(B_i >= 0 && B_i < M)
      {
        Bm[b_y][b_x] = B[B_i];
        Bm[b_y][b_x+1] = (B_i == 0) ?0 :B[B_i-
        1];
      }
      else
      {
        Bm[b_y][b_x] = 0;
        Bm[b_y][b_x+1] = (B_i == M) ?B[B_i-
        1] :0;
      }
    }
  …
}
```

3 本文算法分析

3.1 算法的扩展性

依据式(4),只要将算法表达式可以表示为Toeplitz矩阵与稠密矩阵相乘的这种形式,不仅仅FIR数字滤波,其他的信号处理或者图像处理算法也能适应本文的GPU HsFIR算法.对于一些特殊的线性方程组求解和一维离散卷积的问题,则可以直接应用GPU HsFIR算法.如果将一维离散卷积的形式表示为

$$y = h * x = \mathrm{H} \times \mathrm{X} = \begin{bmatrix} h_0 & 0 & 0 & 0 & \cdots & 0 \\ h_1 & h_0 & 0 & 0 & 0 & \cdots \\ \cdots & h_1 & h_0 & 0 & 0 & 0 \\ h_m & \cdots & h_1 & h_0 & 0 & 0 \\ \cdots & h_m & \cdots & h_1 & h_0 & 0 \\ 0 & \cdots & h_m & \cdots & h_1 & h_0 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ \cdots \\ x_n \end{bmatrix} \quad (6)$$

式(6)即为GPU HsFIR算法在通道数$L = 1$的特殊形式.从数学角度讲,自相关、互相关和移动平均算法都能转化成卷积的形式,也能利用GPU HsFIR算法进行求解.对于倍频程、分数倍频程等基于滤波器组进行求解的算法,更是能够直接应用GPU HsFIR算法.

如果将式(5)的B_M向量定义为

$$\mathrm{B}_M = (b_{-M}, \cdots, b_{-3}, b_{-2}, b_{-1}, b_0, b_1, b_2, b_3, \cdots, b_M),$$

考虑到算法中序号计数从0开始,可将其修改为

$$\mathrm{B}_M = (b_0, \cdots, b_{M-3}, b_{M-2}, b_{M-1}, b_{M+0}, b_{M+1}, b_{M+2}, b_{M+3}, \cdots, b_{2M}),$$

则式(5)就可以写成如下的通用形式

$$b_{ij} = \mathrm{B}_M(i - j + M).$$

对于所有能表示成为Toeplitz矩阵与稠密矩阵相乘形式的算法,GPU HsFIR算法稍加修改后都能够适应.此时算法有2个需要改动的地方:一是从CPU拷贝到GPU常数存储器的数据长度增大了2倍,而GPU常数存储器的长度一般小于64 K;二是内存映射分支的代码,需要修改为适应式(5)的形式.显然上述修改降低了线程之间的分支数量,但是却增加了常数存储器的访问频次,这对算法性能有一定影响.

3.2 算法性能分析

分析GPU HsFIR算法的性能,定义如下的性能指标[15]:

1) 浮点运算能力O_p.单位是Gflop/s,可以用来衡量算法的时间复杂度[19],若设T_p(单位:ms)表示GPU并行执行时间,则O_p定义为

$$O_p = 2.0 \times \frac{N \times N \times L}{T_p \times 10^{-6}}.$$

2) 效率E.算法浮点运算能力与峰值浮点运算能力的比率,用来衡量算法的资源利用能力.设O表示峰值浮点运算能力,则定义

$$E = \frac{O_p}{O} \times 100\%,$$

其中O的取值来自实际测试的最大值和其他文献测试的数值[13-14],具体数值如表1所示.

表1 测试中使用的处理器列表

	核的数量	核心频率/GHz	内存规格	峰值Gflop/s
Intel PD945	2	3.4	DDR2	24 *
Intel Q8200	4	2.33	DDR3	34 *
GeForce 9800GT	112	0.6	DDR3	370
GeForce GTX260+	216	0.625	DDR3	512

注:* Whetstone iSSE3

3) 加速比S.指GPU上求解某问题的并行执行时间与CPU求解同一问题所花时间之比,若定义CPU求解问题的时间为T_s,则$S = \frac{T_s}{T_p}$.此时考虑的仅是GPU上并行执行的时间,而不是在GPU上求解该问题的总开销时间,即S表示并行加速比.实际上,CPU到GPU数据传输的时间在算法总开销时间中占有一定比重,不能忽略,特别是在数据量较小的时候.如图4所示的实验,在$N = 128$时,数据传输时间超过了GPU并行执行时间,占到了整个算法时间的58.53%.但是,由于CPU—GPU数据传输时间主要决定于总线速率和内存延迟,在同类算法中的数值都是固定的,不是算法分析考虑的

RT Embedded http://www.kontronn.com

主要因素,因此后面算法分析中的加速比都采用并行加速比,不再单独分析 CPU —GPU 的数据传输时间.

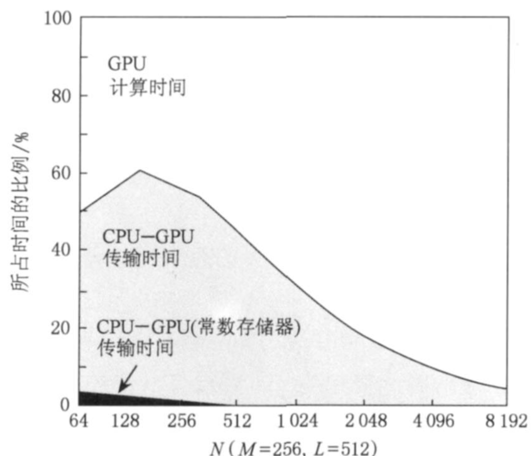


图 4　GPU HsFIR 算法执行时间的细分(GTX260 + ,PCIe2.0)

GPU HsFIR 算法的性能提升是非常显著的,如图 5 所示,在 Intel Q8200+ GTX 260 + 平台上 GPU HsFIR 算法的效率明显超过其他算法,并且随着激励信号长度 N 的增加而提高. 但是这个趋势逐渐放缓,在 $N \approx 4096$ 时效率最高(43.88 %),此时浮点运算能力达到了 219.38 Gflop/s.

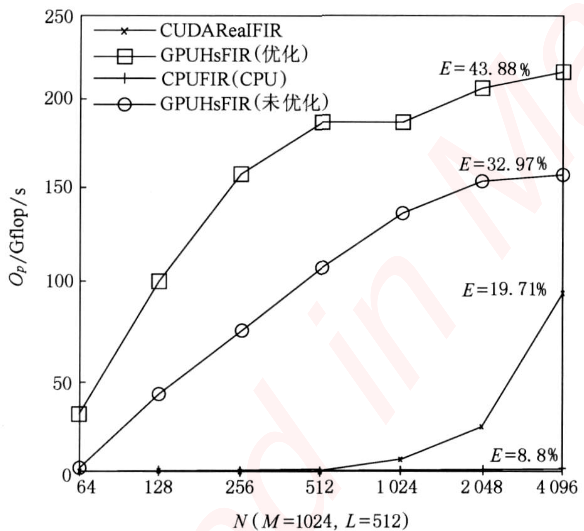


图 5　GPU HsFIR 算法与其他算法的比较

但是,图 6 显示的加速比却在 $N \approx 256$ 时达到 $S \approx 419.51$ 后急剧下降,这也验证了图 5 的结论,当 $N > 4096$ 时,GPU HsFIR 算法的浮点运算能力不会继续提高反而出现下降趋势,而 CPU 求解问题的算法性能却逐渐提高.

分析这个现象需要借助 GPU 的仿真器[20]和 CUDA Visual Profiler(V2.3)工具①对 GPU HsFIR 算法进行负载分析,从而了解算法各执行单元对于

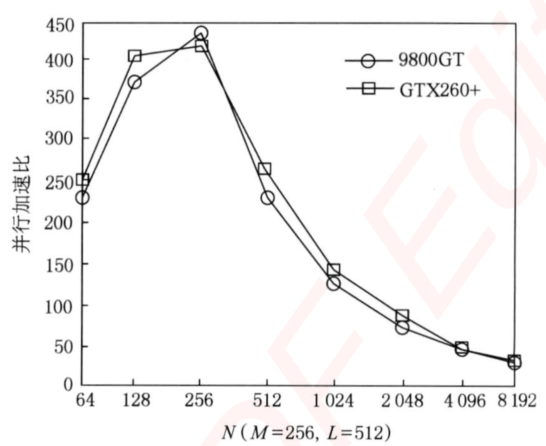


图 6　GPU HsFIR 算法的加速比随 N 增大而变小

其性能的影响作用. 如图 7 所示,虽然算法中的指令执行时间(instructions)和全局存储器操作(gld 128 b,同时读取 2 个数)分别占到 79.09 %和 14.12 %,但是算法分支(branch)和序列冲突(warp serialize)仍然占有一定的比重,随着 N 的增大,这些因素影响算法性能的比重逐渐增加,包括全局存储器操作时间的增加,当 $N > 4096$ 时,算法中指令执行时间的比重是下降的. 这个结论说明:对于执行较大尺度的 FIR 滤波,可以利用重叠块的方法[1]将 N 划分为子片段后再进行滤波,这样能获得最佳的加速性能.

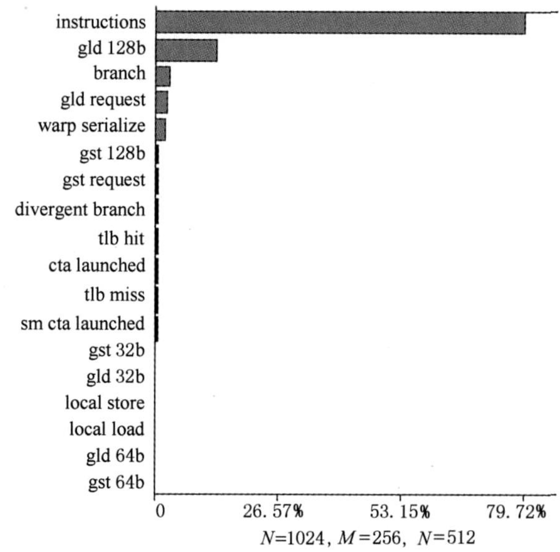


图 7　GPU HsFIR 算法的负载分析

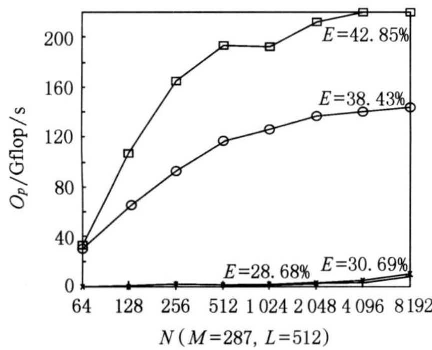
4　实例及分析

本节首先展示 3 种 FIR 数字滤波器的实际例子,

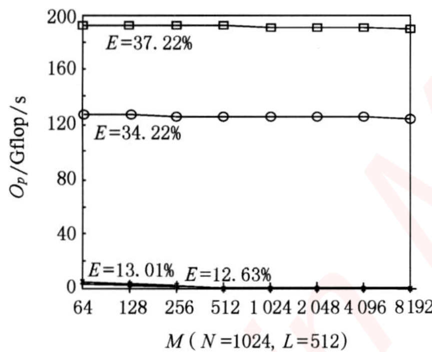
Created in Master PDF Editor

RT Embedded http://www.kontronn.com

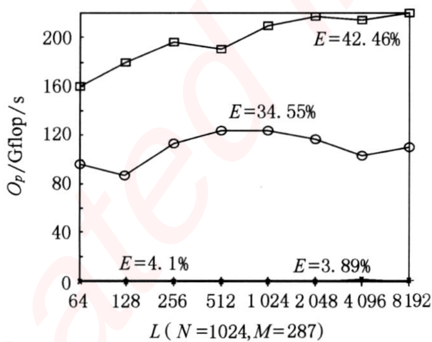
并且依据这些实例在 4 种平台上测试 GPU HsFIR 算法的滤波性能,展现其在 GPU 上的并行扩展能力;其次,通过 FIR 数字滤波的实例,比较 GPU HsFIR 算法与已有 GPU 上 FIR 算法的滤波性能,展现该算法在实际滤波过程的优势. 如图 8 所示,图 8 a～8 c 的激励信号采用 Matlab 生成的白噪声信号,而滤波器设计采用 Orfanidis 提供的 Kaiser 窗函数代码[1]. 图 8 a,8 c 选用低通数字滤波,采样率为 20 kHz,通带频率为 4 kHz,阻带频率为 4.4 kHz,通带纹波为 0.1 dB,阻带衰减为 90 dB,滤波器长度 $M = 287$. 图 8 c 选用 M 为偶数,仅适合低通与带通滤波器的例子.



a 算法性能随激励信号长度 N 变化



b 算法性能随滤波器长度 M 变化



c 算法性能随通道数量 L 变化

注: —○— 9800GT; —□— GTX260+; —+— Intel PD945; —*— Intel Q8200

图 8　采用 GPU HsFIR 算法的滤波器实例在不同平台的性能比较

这里不再展示滤波前后的效果对比,而主要关注算法的滤波性能. 图 8 展示了随着 FIR 数字滤波器 3 个关键参数变化时,GPU HsFIR 算法性能的变化. 显然,GPU HsFIR 算法主要是受到激励信号长度 N 的影响(如图 8 a 所示),而滤波器长度 M 和通道数量 L 的影响并不明显. 另外,从图 8 还可以得知,GPU HsFIR 算法在不同 GPU 平台的加速性能是一致的:在 Intel PD945 + 9800 GT 平台,效率是 38.43 %,加速比平均是 190,最高是 413;而在 Intel Q8200 + GTX260 平台,效率是 42.85 %,加速比平均是 203,最高是 409.

表 2 所示为在 Intel Q8200 + GTX260 平台上,GPU HsFIR 算法与 CUDA RealFIR 算法的滤波性能比较. 由于 CUDA RealFIR 算法[12]在滤波参数上的限制,滤波器实例仍然选用低通数字滤波,采样率为 44.1 kHz,通带频率为 10 kHz,阻带频率为 10.2 kHz,通带纹波为 0.1 dB,阻带衰减为 90 dB,滤波器长度 $M = 1300$ 通道数量 $L = 512$. 表 2 的实例滤波结果显示,本文的 GPU HsFIR 算法的滤波速度已经全面超越了 CUDA RealFIR 算法,但是这 2 种算法的滤波速度差距随着激励信号的长度(N)增大而逐渐变小. 当然,上文提到的 CUDA RealFIR 算法的滤波限制不会随着这个趋势而改变.

表 2　GPU HsFIR 算法与 CUDA RealFIR 算法的滤波性能比较

Gflop/s

	N/O_p	CUDA RealFIR 算法	GPU HsFIR 算法
$M = 1300$ $L = 512$	64	0.05	33.26
	128	0.13	106.85
	256	0.52	173.68
	512	1.72	197.73
	1 024	6.61	196.39
	2 048	26.89	215.64
	4 096	99.56	220.38

5　结　　论

通过研究现有 FIR 数字滤波器在 GPU 实现的算法原理,借鉴其他平台 FIR 优化算法的思想,本文提出了一种在 GPU 上实现 FIR 数字滤波器的高速并行算法 ——GPU HsFIR. 该算法利用矩阵乘法结构在 GPU 实现 FIR 数字滤波,并且优化了并行计算模型和内存映射策略,不仅消除了现有 GPU 实现 FIR 算法的限制,而且获得了较好的算法性能提升.

Created in Master PDF Editor

RT Embedded http://www.kontronn.com

通过 FIR 数字滤波的实例测试,在 Intel Q8200 + GTX260 平台上获得了最高 409 倍的加速比,平均加速比也达到了 203 倍,效率提高到 42.85 %. 另外,该算法具有很好的扩展性,对 FIR 滤波的各项参数不再限制,不仅能够适应更普遍的 FIR 数字滤波(如低通、高通、带通和带阻)的要求,而且能够应用于其他信号处理和图像处理,比如离散卷积、移动平均和分数倍频程算法等.

通过算法性能的分析得知,GPU HsFIR 算法的性能有一定起伏,在激励信号长度是 4 096 时性能最好. 虽然此时加速比不高(50 倍),但是 GPU 的利用效率是最高的,而且滤波速度也是最快的,达到了如图 5 所示的 219.38 Gflop/s (Intel Q8200 + GTX 260 +). 因此,对于执行较大尺度的 FIR 数字滤波,建议利用重叠块的方法[1]划分为子片段(4 096 点)后再进行滤波,这样能使算法具有最佳性能,而且能够充分利用 GPU 并行计算资源.

致谢 感谢 Koon 提供的 CUDA Real FIR 源代码用来测试!感谢中国科学院超级计算中心提供的 GPU 测试环境!感谢北京大学信息科学技术学院陈一峯研究员对于矩阵乘法优化方法的指导!

参考文献(References):

[1] Sophocles J O. Introduction to signal processing (photocopy version) [M]. Beijing: Tsinghua University Press, 1998: 125-567

[2] Park J, Muhammad K, Roy K. High-performance FIR filter design based on sharing multiplication [J]. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 2003, 11 (2): 244-253

[3] Voronenko Y, Püschel M. Multiplierless multiple constant multiplication [J]. ACM Transactions on Algorithms, 2007, 3(2): 11-20

[4] Mahesh R, Vinod A P. A new common subexpression elimination algorithm for realizing low-complexity higher order digital filters [J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2008, 27 (2): 217-229

[5] Li Ying, Lu Weijun, Yu Dunshan, *et al*. A resource optimizing algorithm in FPGA based high speed FIR digital filters [J]. Aeta Seientiarum Naturalium Universitatis Pekinensis, 2009, 45(2): 222-226 (in Chinese)
(李 莹, 路卫军, 于敦山, 等. 一种在 FPGA 上实现 FIR 数字滤波器的资源优化算法[J]. 北京大学学报: 自然科学版, 2009, 45(2): 222-226)

[6] Mou Z J, Duhamel P. Fast FIR filtering: algorithms and implementations [J]. Signal Processing, 1987, 13(4): 377-384

[7] Cheng C, Parhi K K. Hardware efficient fast parallel FIR filter structures based on iterated short convolution [J]. IEEE Transactions on Circuits and Systems I: Regular Papers, 2004, 51(8): 1492-1500

[8] Conway R. Efficient residue arithmetic based parallel fixed coefficient FIR filters [C] //Proceedings of IEEE International Symposium on Circuits and Systems, Seattle: The Printing House, 2008: 1484-1487

[9] Cheng C, Parhi K K. Low-cost parallel FIR filter structures with 2-stage parallelism [J]. IEEE Transactions on Circuits and Systems I: Regular Papers, 2007, 54(2): 280-290

[10] Wu Enhua, Liu Youquan. General purpose computing on GPU [J]. Journal of Computer-Aided Design & Computer Graphics, 2004, 16(5): 601-612 (in Chinese)
(吴恩华, 柳有权. 基于图形处理器(GPU)的通用计算[J]. 计算机辅助设计与图形学学报, 2004, 16(5): 601-612)

[11] Smirnov A, Chiueh T C. An Implementation of a FIR filter on a GPU [R]. New York: State University of New York. Experimental Computer Systems Lab, 2005

[12] Koon L. CUDA real FIR crossover on GPU [OL]. [2009-11-03]. http://koonlab.com/CUDA_RealFIR/CUDA%20Real%20FIR.html

[13] NVIDIA CUDA C programming best practices guide [M]. Santa Clara: NVIDIA Corporation, 2009

[14] Volkv V, Demmel J W. Benchmarking GPUs to tune dense linear algebra [C] //Proceedings of International Conference for High Performance Computing, Networking, Storage and Analysis, Austin: IEEE Press, 2008: 1-11

[15] Grama A, Gupta A, Karypis G, *et al*. Introduction to parallel computing [M]. 2nd ed. Upper Saddle River: Addison Wesley, 2003: 143-272

[16] Gongáeg P, Cabaleiro J C, Pena T F. On parallel solvers for sparse triangular systems [J]. Journal of Systems Architecture, 2000, 46(8): 675-685

[17] Kung S Y, Hu Y H. A highly concurrent algorithm and pipelined architecture for solving Toeplitz systems [J]. IEEE Transactions on Acoustics, Speech, and Signal Processing 1983, ASSP-31(1): 66-76

[18] Sundaram N, Raghunatnan A, Chakradhar S T. A framework for efficient and scalable execution of domain-specific templates on GPUs [C] //Proceedings of the 23rd IEEE International Parallel and Distributed Processing Symposium. New York: IEEE Press, 2009: 1-12

[19] Cormen T H, Leiserson C E, Rivest R L, *et al*. Introduction to algorithms [M]. 2nd ed. Boston: The MIT Press, 2001: 323-331

[20] Bakhoda A, Yuan G L, Fung W W L, *et al*. Analyzing CUDA workloads using a detailed GPU simulator [C] // Proceedings of IEEE International Symposium on Performance Analysis of Systems and Software. New York: IEEE Press, 2009: 163-174

Created in Master PDF Editor

嵌入式资源免费下载

总线协议：

1. 基于 PCIe 驱动程序的数据传输卡 DMA 传输
2. 基于 PCIe 总线协议的设备驱动开发
3. CANopen 协议介绍
4. 基于 PXI 总线 RS422 数据通信卡 WDM 驱动程序设计
5. FPGA 实现 PCIe 总线 DMA 设计
6. PCI Express 协议实现与验证
7. VPX 总线技术及其实现
8. 基于 Xilinx FPGA 的 PCIE 接口实现
9. 基于 PCI 总线的 GPS 授时卡设计
10. 基于 CPCI 标准的 6U 信号处理平台的设计
11. USB30 电路保护
12. USB30 协议分析与框架设计
13. USB 30 中的 CRC 校验原理及实现
14. 基于 CPLD 的 UART 设计
15. IPMI 在 VPX 系统中的应用与设计
16. 基于 CPCI 总线的 PMC 载板设计
17. 基于 VPX 总线的工件台运动控制系统研究与开发
18. PCI Express 流控机制的研究与实现
19. UART16C554 的设计
20. 基于 VPX 的高性能计算机设计
21. 基于 CAN 总线技术的嵌入式网关设计
22. Visual C 串行通讯控件使用方法与技巧的研究
23. IEEE1588 精密时钟同步关键技术研究
24. GPS 信号发生器射频模块的一种实现方案
25. 基于 CPCI 接口的视频采集卡的设计
26. 基于 VPX 的 3U 信号处理平台的设计
27. 基于 PCI Express 总线 1394b 网络传输系统 WDM 驱动设计
28. AT89C52 单片机与 ARINC429 航空总线接口设计
29. 基于 CPCI 总线多 DSP 系统的高速主机接口设计
30. 总线协议中的 CRC 及其在 SATA 通信技术中的应用
31. 基于 FPGA 的 SATA 硬盘加解密控制器设计
32. Modbus 协议在串口通讯中的研究及应用
33. 高可用的磁盘阵列 Cache 的设计和实现
34. RAID 阵列中高速 Cache 管理的优化

Created by Master PDF Editor

35. 一种新的基于 RAID 的 CACHE 技术研究与实现
36. 基于 PCIE-104 总线的高速数据接口设计
37. 基于 VPX 标准的 RapidIO 交换和 Flash 存储模块设计
38. 北斗卫星系统在海洋工程中的应用
39. 北斗卫星系统在远洋船舶上应用的研究
40. 基于 CPCI 总线的红外实时信号处理系统
41. 硬件实现 RAID 与软件实现 RAID 的比较
42. 基于 PCI Express 总线系统的热插拔设计
43. 基于 RAID5 的磁盘阵列 Cache 的研究与实现
44. 基于 PCI 总线的 MPEG2 码流播放卡驱动程序开发
45. 基于磁盘异或引擎的 RAID5 小写性能优化
46. 基于 IEEE1588 的时钟同步技术研究
47. 基于 Davinci 平台的 SD 卡读写优化
48. 基于 PCI 总线的图像处理及传输系统的设计
49. 串口和以太网通信技术在油液在线监测系统中的应用
50. USB30 数据传输协议分析及实现
51. IEEE 1588 协议在工业以太网中的实现
52. 基于 USB30 的设备自定义请求实现方法
53. IEEE1588 协议在网络测控系统中的应用
54. USB30 物理层中弹性缓冲的设计与实现
55. USB30 的高速信息传输瓶颈研究
56. 基于 IPv6 的 UDP 通信的实现

VxWorks：

1. 基于 VxWorks 的多任务程序设计
2. 基于 VxWorks 的数据采集存储装置设计
3. Flash 文件系统分析及其在 VxWorks 中的实现
4. VxWorks 多任务编程中的异常研究
5. VxWorks 应用技巧两例
6. 一种基于 VxWorks 的飞行仿真实时管理系统
7. 在 VxWorks 系统中使用 TrueType 字库
8. 基于 FreeType 的 VxWorks 中文显示方案
9. 基于 Tilcon 的 VxWorks 简单动画开发
10. 基于 Tilcon 的某武器显控系统界面设计
11. 基于 Tilcon 的综合导航信息处理装置界面设计
12. VxWorks 的内存配置和管理
13. 基于 VxWorks 系统的 PCI 配置与应用
14. 基于 MPC8270 的 VxWorks BSP 的移植

Created by Master PDF Editor

15. Bootrom 功能改进经验谈
16. 基于 VxWorks 嵌入式系统的中文平台研究与实现
17. VxBus 的 A429 接口驱动
18. 基于 VxBus 和 MPC8569E 千兆网驱动开发和实现
19. 一种基于 vxBus 的 PPC 与 FPGA 高速互联的驱动设计方法
20. 基于 VxBus 的设备驱动开发
21. 基于 VxBus 的驱动程序架构分析
22. 基于 VxBus 的高速数据采集卡驱动程序开发
23. Vxworks 下的冗余 CAN 通讯模块设计
24. WindML 工业平台下开发 S1d13506 驱动及显示功能的实现
25. WindML 中 Mesa 的应用
26. VxWorks 下图形用户界面开发中双缓冲技术应用
27. VxWorks 上的一种 GUI 系统的设计与实现
28. VxWorks 环境下 socket 的实现
29. VxWorks 的 WindML 图形界面程序的框架分析
30. VxWorks 实时操作系统及其在 PC104 下以太网编程的应用
31. 实时操作系统任务调度策略的研究与设计
32. 军事指挥系统中 VxWorks 下汉字显示技术
33. 基于 VxWorks 实时控制系统中文交互界面开发平台
34. 基于 VxWorks 操作系统的 WindML 图形操控界面实现方法
35. 基于 GPU FPGA 芯片原型的 VxWorks 下驱动软件开发
36. VxWorks 下的多串口卡设计
37. VxWorks 内存管理机制的研究
38. T9 输入法在 Tilcon 下的实现

Linux:

1. Linux 程序设计第三版及源代码
2. NAND FLASH 文件系统的设计与实现
3. 多通道串行通信设备的 Linux 驱动程序实现
4. Zsh 开发指南-数组
5. 常用 GDB 命令中文速览
6. 嵌入式 C 进阶之道
7. Linux 串口编程实例
8. 基于 Yocto Project 的嵌入式应用设计
9. Android 应用的反编译
10. 基于 Android 行为的加密应用系统研究
11. 嵌入式 Linux 系统移植步步通
12. 嵌入式 CC++语言精华文章集锦

Created with Master PDF Editor

13. 基于 Linux 的高性能服务器端的设计与研究
14. S3C6410 移植 Android 内核
15. Android 开发指南中文版
16. 图解 Linux 操作系统架构设计与实现原理（第二版）
17. 如何在 Ubuntu 和 Linux Mint 下轻松升级 Linux 内核
18. Android 简单 mp3 播放器源码
19. 嵌入式 Linux 系统实时性的研究
20. Android 嵌入式系统架构及内核浅析
21. 基于嵌入式 Linux 操作系统内核实时性的改进方法研究
22. Linux TCP IP 协议详解
23. Linux 桌面环境下内存去重技术的研究与实现
24. 掌握 Android 7.0 新增特性 Quick Settings
25. Android 应用逆向分析方法研究
26. Android 操作系统的课程教学
27. Android 智能手机操作系统的研究
28. Android 英文朗读功能的实现
29. 基于 Yocto 订制嵌入式 Linux 发行版
30. 基于嵌入式 Linux 的网络设备驱动设计与实现
31. 如何高效学习嵌入式
32. 基于 Android 平台的 GPS 定位系统的设计与实现
33. LINUX ARM 下的 USB 驱动开发
34. Linux 下基于 I2C 协议的 RTC 驱动开发
35. 嵌入式下 Linux 系统设备驱动程序的开发
36. 基于嵌入式 Linux 的 SD 卡驱动程序的设计与实现
37. Linux 系统中进程调度策略
38. 嵌入式 Linux 实时性方法
39. 基于实时 Linux 计算机联锁系统实时性分析与改进
40. 基于嵌入式 Linux 下的 USB30 驱动程序开发方法研究
41. Android 手机应用开发之音乐资源播放器
42. Linux 下以太网的 IPv6 隧道技术的实现
43. Research and design of mobile learning platform based on Android
44. 基于 linux 和 Qt 的串口通信调试器调的设计及应用
45. 在 Linux 平台上基于 QT 的动态图像采集系统的设计
46. 基于 Android 平台的医护查房系统的研究与设计

Windows CE：

1. Windows CE.NET 下 YAFFS 文件系统 NAND Flash 驱动程序设计
2. Windows CE 的 CAN 总线驱动程序设计

Created with Master PDF Editor

3. 基于 Windows CE.NET 的 ADC 驱动程序实现与应用的研究
4. 基于 Windows CE.NET 平台的串行通信实现
5. 基于 Windows CE.NET 下的 GPRS 模块的研究与开发
6. win2k 下 NTFS 分区用 ntldr 加载进 dos 源代码
7. Windows 下的 USB 设备驱动程序开发
8. WinCE 的大容量程控数据传输解决方案设计
9. WinCE6.0 安装开发详解
10. DOS 下仿 Windows 的自带计算器程序 C 源码
11. G726 局域网语音通话程序和源代码
12. WinCE 主板加载第三方驱动程序的方法
13. WinCE 下的注册表编辑程序和源代码
14. WinCE 串口通信源代码
15. WINCE 的 SD 卡程序[可实现读写的源码]
16. 基于 WinCE 的 BootLoader 研究
17. Windows CE 环境下无线网卡的自动安装
18. 基于 Windows CE 的可视电话的研究与实现
19. 基于 WinCE 的嵌入式图像采集系统设计
20. 基于 ARM 与 WinCE 的掌纹鉴别系统
21. DCOM 协议在网络冗余环境下的应用
22. Windows XP Embedded 在变电站通信管理机中的应用
23. XPE 在多功能显控台上的开发与应用
24. 基于 Windows XP Embedded 的 LKJ2000 仿真系统设计与实现

PowerPC：

1. Freescale MPC8536 开发板原理图
2. 基于 MPC8548E 的固件设计
3. 基于 MPC8548E 的嵌入式数据处理系统设计
4. 基于 PowerPC 嵌入式网络通信平台的实现
5. PowerPC 在车辆显控系统中的应用
6. 基于 PowerPC 的单板计算机的设计
7. 用 PowerPC860 实现 FPGA 配置
8. 基于 MPC8247 嵌入式电力交换系统的设计与实现
9. 基于设备树的 MPC8247 嵌入式 Linux 系统开发
10. 基于 MPC8313E 嵌入式系统 UBoot 的移植
11. 基于 PowerPC 处理器 SMP 系统的 UBoot 移植
12. 基于 PowerPC 双核处理器嵌入式系统 UBoot 移植
13. 基于 PowerPC 的雷达通用处理机设计

Created with Master PDF Editor

14. PowerPC 平台引导加载程序的移植
15. 基于 PowerPC 嵌入式内核的多串口通信扩展设计
16. 基于 PowerPC 的多网口系统抗干扰设计
17. 基于 MPC860T 与 VxWorks 的图形界面设计
18. 基于 MPC8260 处理器的 PPMC 系统
19. 基于 PowerPC 的控制器研究与设计
20. 基于 PowerPC 的模拟量输入接口扩展
21. 基于 PowerPC 的车载通信系统设计

ARM：

1. 基于 DiskOnChip 2000 的驱动程序设计及应用
2. 基于 ARM 体系的 PC-104 总线设计
3. 基于 ARM 的嵌入式系统中断处理机制研究
4. 设计 ARM 的中断处理
5. 基于 ARM 的数据采集系统并行总线的驱动设计
6. S3C2410 下的 TFT LCD 驱动源码
7. STM32 SD 卡移植 FATFS 文件系统源码
8. STM32 ADC 多通道源码
9. ARM Linux 在 EP7312 上的移植
10. ARM 经典 300 问
11. 基于 S5PV210 的频谱监测设备嵌入式系统设计与实现
12. Uboot 中 start.S 源码的指令级的详尽解析
13. 基于 ARM9 的嵌入式 Zigbee 网关设计与实现
14. 基于 S3C6410 处理器的嵌入式 Linux 系统移植
15. CortexA8 平台的μC-OSⅡ及 LwIP 协议栈的移植与实现
16. 基于 ARM 的嵌入式 Linux 无线网卡设备驱动设计
17. ARM S3C2440 Linux ADC 驱动
18. ARM S3C2440 Linux 触摸屏驱动
19. Linux 和 Cortex-A8 的视频处理及数字微波传输系统设计
20. Nand Flash 启动模式下的 Uboot 移植
21. 基于 ARM 处理器的 UART 设计
22. ARM CortexM3 处理器故障的分析与处理
23. ARM 微处理器启动和调试浅析
24. 基于 ARM 系统下映像文件的执行与中断运行机制的实现
25. 中断调用方式的 ARM 二次开发接口设计
26. ARM11 嵌入式系统 Linux 下 LCD 的驱动设计
27. Uboot 在 S3C2440 上的移植

Created with Master PDF Editor

28. 基于 ARM11 的嵌入式无线视频终端的设计
29. 基于 S3C6410 的 Uboot 分析与移植
30. 基于 ARM 嵌入式系统的高保真无损音乐播放器设计

Hardware：

1. DSP 电源的典型设计
2. 高频脉冲电源设计
3. 电源的综合保护设计
4. 任意波形电源的设计
5. 高速 PCB 信号完整性分析及应用
6. DM642 高速图像采集系统的电磁干扰设计
7. 使用 COMExpress Nano 工控板实现 IP 调度设备
8. 基于 COM Express 架构的数据记录仪的设计与实现
9. 基于 COM Express 的信号系统逻辑运算单元设计
10. 基于 COM Express 的回波预处理模块设计
11. 基于 X86 平台的简单多任务内核的分析与实现
12. 基于 UEFI Shell 的 PreOS Application 的开发与研究
13. 基于 UEFI 固件的恶意代码防范技术研究
14. MIPS 架构计算机平台的支持固件研究
15. 基于 UEFI 固件的攻击验证技术研究
16. 基于 UEFI 的 Application 和 Driver 的分析与开发
17. 基于 UEFI 的可信 BIOS 研究与实现
18. 基于 UEFI 的国产计算机平台 BIOS 研究
19. 基于 UEFI 的安全模块设计分析
20. 基于 FPGA NiosⅡ的等精度频率计设计
21. 基于 FPGA 的 SOPC 设计
22. 基于 SOPC 基本信号产生器的设计与实现
23. 基于龙芯平台的 PMON 研究与开发
24. 基于 X86 平台的嵌入式 BIOS 可配置设计
25. 基于龙芯 2F 架构的 PMON 分析与优化
26. CPU 与 GPU 之间接口电路的设计与实现
27. 基于龙芯 1A 平台的 PMON 源码编译和启动分析
28. 基于 PC104 工控机的嵌入式直流监控装置的设计
29. GPGPU 技术研究与发展

Created with Master PDF Editor

Programming:

1. 计算机软件基础数据结构 - 算法
2. 高级数据结构对算法的优化
3. 零基础学算法
4. Linux 环境下基于 TCP 的 Socket 编程浅析
5. Linux 环境下基于 UDP 的 socket 编程浅析
6. 基于 Socket 的网络编程技术及其实现
7. 数据结构考题 - 第 1 章 绪论
8. 数据结构考题 - 第 2 章 线性表
9. 数据结构考题 - 第 2 章 线性表 - 答案

Created in Master PDF Editor